



Montana Department of DOT

HEAT Application Design Recommendations

Prepared By: Esri
Prepared For: Miles Wacker, GISP
Geospatial Systems Analyst
Information Services Division
Montana Department of
Transportation
(406) 444-0414

Last Modification Date: April 1, 2013



esri®

380 New York Street
Redlands, California 92373-8100 USA
T 909 793 2853



Revision History

Date	Description	Person Responsible
06/20/2012	Baseline document	Esri
10/30/2012	Final Released	Esri



Table of Contents

1.	About This Document	5
1.1	Purpose	5
1.2	Scope.....	5
1.3	Target Audience	5
1.4	Authors & Participants	5
1.5	Assumptions.....	5
1.6	Related Documents.....	5
2	Design Recommendations	6
2.1	Current Script Migration	6
2.2	Application Architecture	6
2.3	Network Data	6
2.3.1	No Build Scenario	6
2.3.2	Scenarios	9
3	Data Sources	11
3.2	Script Analysis	13
3.2.1	Assignment.....	13
3.2.2	Calculate Population & Employment 60 Minute Time to Travel	14
3.2.3	Calculate Maximum Potential Growth.....	14
3.2.4	Get Population & Employment in Net Bands.....	14
3.2.5	Calculate Accessibility	15
4	Script Functionality	16
4.1	Assignment.ave Avenue Script Functionality.....	16
4.2	Calc_pop_emp_60mintt.ave.....	20
4.3	Calculats_max_pot_gro_final.ave.....	21
4.4	Gets_popandemp_innetbands.ave.....	22
4.5	Calcs_acs.ave	23
4.6	Calcs_dgo.ave.....	24
4.7	Cao_ave.....	24
4.8	Accessibility Avenue Script Functionality.....	27





1. About This Document

1.1 Purpose

This document investigates design options for the Montana Department of DOT [MTDOT] HEAT Upgrade project. The purpose of this document is to present design options for upgrading the HEAT application to a modern software platform.

1.2 Scope

The scope of design options specified by this document includes the HEAT Application and its related analysis modules owned by MTDOT. Third party analysis modules were not analyzed in detail.

1.3 Target Audience

The target audience of this document includes the MTDOT technical staff responsible for developing a detailed design specification for the new HEAT application.

1.4 Authors & Participants

This document is a collaborative effort between Esri Implementation Services and MTDOT teams.

1.5 Assumptions

The following assumptions have been made in writing this document.

ArcGIS version 10.1 systems and functionality will be used.

1.6 Related Documents

Related documents include reference materials from which this document has derived some of its content. In addition, this document may refer to procedures and information that is maintained external to this document.

Document Name	Date	Document Owner



2 Design Recommendations

2.1 Current Script Migration

The current application is composed of several Avenue scripts that were developed almost 10 years ago. Since the scripts were developed, GIS has changed significantly in the way it is used and functionality of the system. Several aspects of the application and even the scripting language itself have been deprecated. Because of the age of the application, scripts, and underlying technology it is recommended that MTDOT abandon the current script technology and pursue new development. This will allow MTDOT to take advantage of new functionality and adapt the application to fully utilize the power of a modern GIS.

It is not recommended that MTDOT attempt to convert the existing scripts to .NET or other such modern GIS language. This process will take significant time and energy to complete and will result in a system that is still stuck attempting to implement old processes with new technology. MTDOT would be better off designing the new system to take advantage of as much 'out of the box' (OOTB) functionality as they can, to avoid (or at least prolong) pre-mature obsolescence.

2.2 Application Architecture

Montana Department of Transportation HEAT application should be broken into a series of web services that can be called successively by an application to produce the final result. Each of the services should be constructed with an interface that can be agreed to by all stake holders. The interface should seek to be simple to understand and provide the minimum amount of functions and attributes to achieve the desired result. If needed, the complexity of interacting with a particular analysis can be implemented behind the interface using an adapter.

For analyses that are not controlled by the Heat application developers, it is recommended that they introduce a façade wrapper for that analysis in order to abstract their applications from the underlying analysis implementation. The façade can hide the complexity and potential for change in the underlying analysis module from the calling application. If a change does occur in the underlying 3rd party module, the façade will likely be the only object impacted. In this manner, underlying and uncontrolled changes to analysis components can be mitigated to the relatively simple façade.

2.3 Network Data

2.3.1 No Build Scenario

Esri's Network Analyst should be used to construct and maintain the "No Build" [NB] network. In ArcView GIS, a persistent network was created when a network analysis function was run on a line shapefile for the first time. In the modern ArcGIS framework, the "Network Dataset" [ND] stores this persistent network. You can save the network, modify its properties, and model a variety of networks using network datasets.

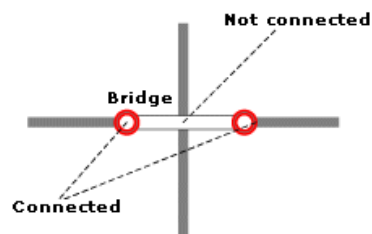
The previous version of HEAT manually calculated network 'costs' based on the origin-destination network. The modern ND calculates these costs on creation, based on the properties defined for the ND.





The NB ND will need to be constructed based on current information contained in several locations available to MTDOT. The NB ND will consist of the following components:

1. The road network – This is the road centerlines that make up the MTDOT network of roads within scope for HEAT. The network should be constructed with the following rules:
 - a. Network segments
 - i. Endpoint Connectivity – Network segments that intersect in the real-world will intersect geometrically. This means the end point of one intersecting segment will be within a specified intersection tolerance of the intersected segment(s).
 - ii. Network segments that do not intersect in the real world will not intersect geometrically. This means the endpoints of two non-intersecting segments will be separated by more than the intersection tolerance.
 - iii. Bridges –
 1. Option 1 - breaking a line before and after an intersection (outside the tolerance) can be used to model a bridge or underpass.



2. Option 2 - Alternatively network segments can use a Z (height) value to indicate intersection or non-intersection of geometrically coincident segments. This means an overpass could have a Z value of 20 and the underpass road could have a Z value of 0. This would allow roadway segments that cross each other in a 2D environment, but do not intersect in a 3D environment to be defined.
 - iv. Network Connectivity Groups – Intermodal connections or connections to other groups of roads can be modeled with connectivity groups. Outside of defining connections to local road networks, it doesn't appear this tactic has immediate applicability to the HEAT paradigm. On first interrogation, it appears connectivity groups could be used to define scenarios. However, the redundancy of network connections between NB and scenario features doesn't fit the intended use of connectivity groups. This will create an overly complex network segment data set that will be hard to visualize and analyze.
- b. The Intersection Set – This is a set of point geometries that identifies where road network segments intersect. You can use this point data set to identify intersection points on the network rather than relying on geometric intersection or Z values as discussed above.
- c. Turns – A feature class that can identify the nature of a specific intersection. For example a left turn may not be allowed at a specified intersection.



2. Attributes – A set of attributes for each road segment that indicate how the segment can be used and implications of that use. Attributes should be used to assign either constriction or cost to the NB set. Within the HEAT context, it appears the Cost Attribute is the most applicable. The Descriptor may also be useful in the [] analysis.

- a. Cost - These indicate certain costs of a road segment. Within the HEAT context, these attributes include travel times, vehicle trips per day, truck trips per day, etc. It should be noted that these cost attributes are calculated values based on descriptors (see below). The calculations are done for each segment when the network is constructed. Further analysis of the network (such as Origin-Destination or Drive Times) will aggregate the cost attributes on each segment in order to solve the problem at hand. In this manner, complex cost analysis can be done on the network without custom code.

- i. Drive Time – A function of vehicle speed and segment length. With historical data, a travel profile based on travel time should be used.

$$([\text{SpeedLimit}]/[\text{SegmentLength}]) \times [\text{Travel Time Factor}(\geq 1)]$$

- ii. Trips per Day – A function of the road capacity and percent auto or truck traffic. With historical data, a travel profile based on travel speed should be used.

$$([\text{Total Capacity}] \times [\% \text{ Auto Traffic}]) \times [(0 \leq) \text{Traffic Speed Factor}(\leq 1)]$$

- iii. Historic Traffic Counts - MTDOT is encouraged to use Historical traffic data to provide traffic profiles to existing NB road networks. This will provide the most accurate NB profiles possible. MTDOT has a unique ability to understand the historical traffic flows on their road networks and they should take advantage of this capability. These counts translate into a factor that is applied against the control cost attribute.

1. Travel Time Profile Type - traffic counts can be used to apply a factor (greater than 1) to the travel time of a segment based on the time of day. A factor of 1 indicates the segment can be travelled at the speed limit for the entire duration of the segment. A factor greater than 1 indicates the travel time will be longer.

Example: Rush hour traffic may cause the actual average speed to be 50% of the speed limit along the entire segment. So the time to travel the segment would double, resulting in a travel time factor of 2.

2. Travel Speed Profile Type – traffic counts can be used to apply a factor (between 0 and 1) to the average speed of vehicles on the segment. A value of 1 indicates that traffic is travelling at the speed limit. A value of less than 1 indicates that traffic speeds will be less than the speed limit.

Example: Rush hour traffic may cause the actual average speed to be 50% of the speed limit along the entire segment. So the actual vehicles traversing a segment may be half the design capacity of the road, resulting in a travel speed factor of 0.50.



- b. Descriptor – These indicate a static attribute along a road, such as the speed limit, number of lanes, or the capacity of a segment. HEAT should use this information in conjunction with travel profiles to derive the network cost attributes.
- c. Restriction – These indicate restrictions on the segment such as a bridge clearance, maximum load, or traffic flow in only one direction. While this attribute has very real applications in DOT over-weight/over-dimension routing, it doesn't appear to have any influence on the MTDOT HEAT application. The one exception here is one-way roads, but initial inspection of the NB dataset didn't indicate these were used.

2.3.2 Scenarios

Scenarios represent reconfigurations of the network. Each scenario will add, edit, or delete segments from the NB to create the new configuration. One of the first questions about scenarios is related to storage. Esri noted the following requirements from MTDOT:

- Scenarios should be persistent and available for future analysis.
- Scenarios should be able to be copied, branched, and/or modified further.
- Scenario data should be easily incorporated into the NB data set if the scenario is implemented.

Based on these requirements, several options were considered:

1. Scenarios as Database Versions – Versioning in a database isolates and tracks the modifications of features within the network dataset. Within ArcGIS, a version of a data set is tracked by keeping a record of all additions, attribute edits, and deletions from its parent data set.
 - a. Pros
 - i. Versions can be created from other versions; which satisfies the copy scenario requirement.
 - ii. Only the changes to the NB network are recorded.
 - iii. The creation of versions is relatively simple and can be scripted.
 - b. Cons
 - i. Versioning can cause the database to become quite large and complex.
 - ii. Multiple version trees (version of a version of a version) can become quite complex and difficult to reconcile.
 - iii. Versions will impair performance as the add/delete tables are traversed.
 - iv. Changes to the underlying NB dataset will become difficult to reconcile with older versions.
 - v. Esri best practices recommend against multiple long term versions.
2. Scenarios as Copies – Traditionally HEAT has maintained copies of projects that include all data associated with that scenario (NB and Scenario data).
 - a. Pros
 - i. Eases maintenance of the NB dataset. It can be maintained by a GIS admin on a regular basis without having to worry about versions that are hanging off of it.
 - ii. Allows the scenario data set to be archived outside the analysis database, reducing database size.



- iii. Eases data handling and computational analysis loads by eliminating the version information.
 - iv. Copying the NB feature classes to a new ND is relatively simple and can be scripted.
- b. Cons
 - i. Depending on how small the scenario changes are, this method can duplicate a significant amount of data between the NB and the scenario.
 - ii. Based on comments from MTDOT, as many as 15 scenarios may be under consideration at any one time. This may create confusion.
- 3. Scenarios As Archives – Scenario data is created either as a version or as a separate dataset. Once the scenario analysis is completed, the data is archived as a historical version. This archive can be accessed at any time, but represents the state of the system (the scenario and possibly the no-build) at the moment in time when the archive version was taken (completed).
 - a. Pros
 - i. Allows the scenario data to be archived outside the analysis database, reducing database size.
 - ii. May be easier to maintain than database backup/replication above.
 - iii. Versions of the NB dataset can still be utilized to model the scenarios, potentially simplifying their management and creation.
 - iv. As data is archived, the scenario version can be removed.
 - b. Cons
 - i. Versions may still exist for prolonged periods of time, complicating the data structure.
 - ii. Management of the archives may become an issue.



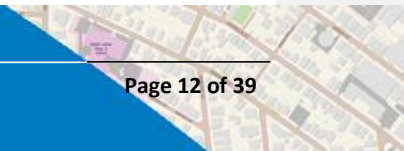
3 Data Sources

The following data sources were identified with comments. Other data sources appear to be provided as hard coded tables that accompany the tools. For the static tables, it may be possible to store them in an relational database where the values can be updated upon change, and calculations can be provided automatically with views instead of code.

1. 2025_TRIPS_OD_ALL_FINAL_TWOWAY – This table appears to contain the traffic count information for trips between nodes.
 - a. Observations:
 - i. It is assumed this table forecasts traffic counts for 2025 (?).
 - ii. The table only accounts for node IDs <= 138 (56 counties, the rest outside of the state).
 - iii. No source for this data was found.
 - b. Recommendations:
 - i. Find documentation for how this information was derived.
 - ii. Establish a process (historical traffic counts + trends) to model future traffic.
2. NOBUILD_NODES – Set of road network nodes that indicate segment connections. Also used to indicate origin and destination nodes.
 - a. Observations:
 - i. Each node is located on an intersection of the road network.
 - ii. NODEID:
 1. Nodes 1-99 (total of 56) correspond to county centers
 - a. Population centers?
 - b. County seats?
 - c. Most are attached to the road network by an artificial road segment.
 - d. They are not the centroid of County polygon.
 2. Nodes 100-999 correspond to roads leaving Montana
 - a. The node and the road they are on are not consistent with actual road shapes (it is not clear how the roads got their shapes).
 - b. If the distance to the node is used in calculations (cost = distance x speed) then these artificial locations may cause inaccuracy.
 3. Nodes 1000+ correspond to road segment intersections.
 - a. These nodes should be managed separately as part of the network dataset.
 - iii. Recommendations
 1. Manage Nodes 1000+ separately as a part of the Network Dataset.
 2. County Nodes
 - a. Manage these nodes as a separate feature class that are inputs to the OD Matrix calculation.



- b. Use the Network Analyst Calculate OD Matrix tool with these points as one of the inputs.
 - c. Snapping these to the road network rather than creating artificial connectors might provide more accurate calculations. The artificial connectors may lead to artificial bottlenecks in the system.
 - d. Another option could be to snap a point on the existing road network that represents the most populated location in the county.
 - 3. State Border Nodes
 - a. It might make more sense to put the nodes on the actual road right at the border.
 - b. This would eliminate any chance of inaccuracy on the exterior road.
- 3. NOBUILD – Set of road segments.
 - a. Observations
 - i. There are a lot of fields on these records that have unknown sources
 - 1. Traffic Flow values (STCC_0, STCC_1, STCC_2, STCC_3, STCC_4, STCC_5, AUTO_ODME, NFT_ODME, TOTAL)
- 4. MON_BLKGRP – The Montana Block Groups.
 - a. Observations
 - i. Census block groups as used/maintained by the state.
 - ii. Population for each block
 - iii. Employment for each block
 - b. Recommendations
 - i. This feature class should be provided by the census authority.
 - ii. This feature class should be maintained in a current state by the census authority.
 - iii. The data should either be utilized directly from the source or replicated into the HEAT database on a regular basis (on change).





3.2 Script Analysis

3.2.1 Assignment

3.2.1.1 Origin-Destination Matrix

In the original application the Assignment process creates a geometric network and calculates an Origin-Destination Matrix (OD Matrix) for a set of specified route nodes. The OD Matrix provides a cost matrix for traversing from one node to another node. The nodes appear to be the centroid of each county or a point outside the state representing a destination outside of the State. The cost is the time to travel along the shortest route between the nodes.

Additional attributes of each route include the following traffic indicators (there is no indication what these measures mean): STCC_0, STCC_1, STCC_2, STCC_3, STCC_4, STCC_5, AUTO_ODME, NFT_ODME, and TOTAL. The field ALL_TRUCK equals TOTAL – AUTO_ODME and appears to indicate truck traffic on the route. The network dataset should be used to calculate all of these attributes for each segment ahead of time. It should be noted that the values of each of these attributes is hard-coded on the segments and their source is unknown to the author.

The original script used to create the OD matrix implemented a very manual process that is no longer applicable to a modern GIS. The network dataset should be used to calculate these values as the ND can be leveraged to automatically manage this data. At its core, the network is defined by segments (lines) and nodes (places where segments intersect). An OD matrix also defines a set of Origin and Destination nodes that (ideally) are co-located on a network node. It is suggested that the origin and destination nodes:

1. Be managed separately from the network nodes – Because these represent places that are start or end of trips, they are conceptually different from the intersection of two segments.
2. Be placed directly on the road network, and not on artificial connectors – Currently it appears that artificial segments were created to connect the origin or destination nodes to the network. These artificial connectors could introduce error into the calculations that rely on distance (e.g. trip time).

Once the script has the OD Matrix, it assigns traffic flow volumes based on the traffic attributes listed above. For each OD Pair, the traffic is added to the segments along the path. The attributes that store this information include: ST0_25, ST1_25, ST2_25, ST3_25, ST4_25, ST5_25, AUTO_25, NFT_25, and TOT_VOL_25. The ND and OD Matrix tool can be used to derive this information as a result of the OD matrix generation.

3.2.1.2 Select Link

In addition to creating the OD Matrix for the set of Origin/Destination nodes, a “Selected Link” table is also created that contains the same details for the selected links in the NOBUILD network. This table summarizes for each link that has been selected, the Origin Node (from), Destination Node (to), Auto volume, Truck volume, From District, To District, and a From/To District (text concatenation of the two). This table indicates, for each selected link, a summary of volumes and districts traveled between.



Finally, a summary table is created for auto and truck traffic with percentage of total traffic for that district that is leaving (based on from district field). First each district pair is summarized (sum data for all values in the link table with identical From/To Districts). Next, the traffic for all selected links are totaled (total selected volumes for truck and auto). These total values are used below in calculating the volume percent for each selected district. Each district's total is then calculated for auto and truck volumes (for each from district). Finally, each district's auto and truck volumes (from the district, flowing out) are calculated as a percentage of the total selected volumes.

DISTRICT	AUTO	TRUCK
1	0.0037	0.2424
2	0.0004	0.2207
3	0.0308	0.0179
4	0.8205	0.3886
5	0.0134	0.1238
99	0.1312	0.0066

It should be noted that the selected link field in the network appears to be a user defined set of links. It is unclear how or when this flag value is set.

3.2.2 Calculate Population & Employment 60 Minute Time to Travel

This tool operates on a "Netband Theme" layer that is assumed to be a drive time (60 minute) buffer around a specified node. The netband layer is selected by the user in the original application. It also uses "Mont_BlckGrp.shp" layer that represents the Montana Block Groups shapes. The netband theme features are intersected with the Montana block group to determine the average population and average employment that lives within the specified feature (prorated based on area within the block multiplied by population or employment and divided by total length of line).

3.2.3 Calculate Maximum Potential Growth

This tool calculates the maximum potential growth for each county. It is based on several tables that appear to be hard coded to the tool. The result is a list of counties with potential growth statistics for each based on the current date. It appears the same thing could be accomplished using a database view based on current (or frequently updated) growth estimates.

3.2.4 Get Population & Employment in Net Bands

Iterate over a set of user provided "net theme" feature classes. Each of these layers appear to represent a set of features for a specified county (county name is embedded within the layer name). For each record in the net theme the population (_00) and workers values are prorated based on the records spatial intersection with the Montana Block Group layer (intersected area multiplied by the population or workers block group value divided by feature total area). The prorated values for each feature are added together to provide an average of the population (Population_00) and workers (Employees) for that feature.



3.2.5 Calculate Accessibility

This tool appears to be calculating its results based on the county name and static tables within the module. It is assumed that the same calculations could be provided by a view on the database based on standard accessibility table that are frequently updated.



4 Script Functionality

4.1 Assignment.ave Avenue Script Functionality

Stacia Taggart
August 6, 2012

The Avenue scripts were built to support Network Analyst functionality that was lacking in the ArcGIS 8.3/9 version of the extension. The Avenue functions were used as part of the following HEAT tools:

- Assignment
- Accessibility

This document describes the functionality in the Assignment.ave script. It appears that the only difference between this script and the ScenAssignment.ave script is that Assignment works on the NOBUILD network, while ScenAssignment works on a specific scenario.

Purpose

The purpose of the Assignment procedure is to assign traffic volumes to the scenario and NOBUILD highway networks based on an origin-destination (OD) table. The assignment uses an “all or nothing” approach, which means that the shortest path from origin to destination is always used, regardless of the volumes on the roadways (a reasonable assumption for most uncongested roads). Any time the NOBUILD and scenario highway attributes are updated, this must be re-run.

Process

Add data to the project

NOBUILD polyline layer
NOBUILDNODES point layer
Trip table (2025_trips_od_all_final_twoway.dbf)

- Delete “view1” if it exists
- Create a new blank “view1”
- Save the project
- Add NOBUILD polyline layer to view
 - The Montana NOBUILD network is used as the basis for each HEAT Scenario. This script will create a Geometric Network of it and store it in ESRI’s Geodatabase format.
 - Location is hard-coded in the script. If the NOBUILD layer is not in “view1”, add it from hard-coded location.
 - Turn the NOBUILD layer on and make it active
- Add NOBUILDNODES point layer to view
 - Contains the nodes for the NOBUILD network. Locations is hard-coded in this script.
 - If the NOBUILDNODES layer is not in view1, add it from hard-coded location



- Turn the NOBUILDNODES layer on
- If the “2025_trips_od_all_final_twoway.dbf” table is not in the project, add it from hard-coded location.
- Save the project

Add Volume Flow Fields to the NOBUILD layer and set them to zero

- If the following fields exist in the NOBUILD layer, zero them out; otherwise, add them to the NOBUILD layer as doubles (20,6):
 - TOT_VOL_25
 - AUTO_25
 - NFT_25
 - ST0_25
 - ST1_25
 - ST2_25
 - ST3_25
 - ST4_25
 - ST5_25

Calculate Travel Time in NOBUILD layer

- If the field TRAVELTIME does not exist in the NOBUILD layer, add it as a double (20,3).
- Calculate the TRAVELTIME field from the “Length” and “Finalspeed” fields:
 - $TRAVELTIME = (LENGTH / FINALSPEED) * 60$

Make the Geometric Network

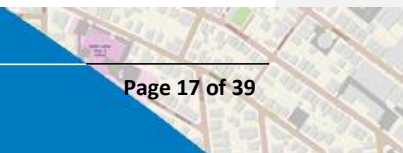
- Clear the feature selection in the NOBUILD layer
- Make a geometric network from the NOBUILD table

Set the Impedance Field to TravelTime

- Set the Cost Field of the geometric network to the TRAVELTIME field

Create Dictionaries for Centroid Connectors

- Select points in the NOBUILDNODES layer that have NODEID < 999.
- Create a dictionary (NodeDistrictDict) of NodeIDs to Districts
- Create a dictionary (NodeDict) of NodeIDs to Point shapes





Calculate Shortest Path for each OD pair in table

- Query records in Trip table where TOTAL field > 0
- Create a new, empty dictionary (LinkDict)
- Create a new, empty List (LinkList)
- Iterate over selected records in Trip table, getting the ROWS and COLS field values. Compare the ROWS and COLS fields and if they are **not equal**,
 - Make a selection set of centroid connectors not to draw a path thru
 - Query the NOBUILD layer
 - Query string: ((([CCTAG] = 0) or ([CCTAG] = " + row.asstring + "))) or ([CCTAG] = " + col.asstring + ")
 - Invert the selection
 - Calculate ALLTRUCK from the TOTAL and AUTO_ODME fields in the Trip table
 - $ALLTRUCK = TOTAL - AUTO_ODME$
 - Get the values of the following fields and add them to a list (FlowList)
 - STCC_0
 - STCC_1
 - STCC_2
 - STCC_3
 - STCC_4
 - STCC_5
 - AUTO_ODME
 - NFT_ODME
 - TOTAL
 - Get the node point from the NodeDict for the value of the ROWS field
 - Get the node point from the NodeDict for the value of the COLS field
 - Make sure the ROWS and COLS points are on the NOBUILD network
 - Find the shortest path on the network between the ROWS point and the COLS point
 - Get the "path shape" as a polyline
 - Use the "path shape" to make a new selection on the NOBUILD polyline layer
 - Iterate over the selected NOBUILD features
 - For each feature, intersect its shape with the "path shape"
 - If the intersection is null or the intersection length divided by the NOBUILD feature length is less than 0.2, continue to the next feature
 - Otherwise, do Select Link Check
 - Get the value of the SELLINK field for the feature
 - If the value = 1, create a list of the ROWS, COLS, AUTO_ODME and ALLTRUCK values and add them to the LinkList
 - If the NodeID key does not exist in the LinkDict, then add it with a value of the FlowList
 - Otherwise, sum the individual values in the existing NodeID's FlowList with the new FlowList created in the Select Link Check

Comment [SLT1]: This selection set doesn't appear to be used



- Clear the network path result
- Clear the NOBUILD polyline feature class selection

Write Flows to Links

- Iterate over all the features in the NOBUILD polyline feature class
 - For each feature, get the ObjectID
 - Use the ObjectID as a key to get the corresponding FlowList from the LinkDict
 - If there is no FlowList, continue
 - Otherwise, set the following NOBUILD field values with those from the FlowList:
 - ST0_25 = FlowList[0]
 - ST1_25 = FlowList[1]
 - ST2_25 = FlowList[2]
 - ST3_25 = FlowList[3]
 - ST4_25 = FlowList[4]
 - ST5_25 = FlowList[5]
 - AUTO_25 = FlowList[6]
 - NFT_25 = FlowList[7]
 - TOT_VOL_25 = FlowList[8]

Comment [SLT2]: Apparently, the NoBuildNodes' NodeID field is related to the NoBuild's ObjectID field. Bad idea.

Write Select Link Table Results

- Create a new table (SelLinkTable) with the following fields:
 - ROW: long
 - COL: long
 - AUTO: double (16,4)
 - TRUCK: double (16,4)
 - F_DIST: long
 - T_DIST: long
 - FTDIST: text (16)
- Iterate over values in LinkList:
 - For each value, add a new row to the table
 - Set the row's field values as follows:
 - ROW = LinkList[0]
 - COL = LinkList[1]
 - AUTO = LinkList[2]
 - TRUCK = LinkList[3]
 - F_DIST = NodeDistrictDict[LinkList[0]]
 - T_DIST = NodeDistrictDict[LinkList[1]]
 - FTDIST = F_DIST + " " + T_DIST



Write Select Link Table Summarize Results

- Summarize the SellLinkTable (SumSelLink) with the following parameters:
 - Summarize field: FTDIST
 - Calc Fields: F_DIST (Min), T_DIST (Min), AUTO (Sum), TRUCK (Sum)
- Sum up all the Auto (AUTOTOT) and Truck (TRUCKTOT) totals
 - Iterate over the results in SumSelLink
 - Sum up all the SUM_AUTO values (AUTOTOT)
 - Sum up all the SUM_TRUCK values (TRUCKTOT)
- Create a new table (SumSelResults) with the following fields:
 - DISTRICT: long
 - AUTO: double (16,4)
 - TRUCK: double (16,4)
- Iterate over a hard-coded list of districts (DistList = 1,2,3,4,5,99)
 - For each district, query the SumSelLink table where MIN_F_DIST = district
 - Add up the Auto and Truck values for that district
 - Iterate over the selected rows
 - Sum up all the SUM_AUTO values (SUMAUTOD)
 - Sum up all the SUM_TRUCK values (SUMTRUCKD)
 - Add a new record to the SumSelResults table and assign the following values:
 - DISTRICT = district
 - AUTO = SUMAUTOD / AUTOTOT
 - TRUCK = SUMTRUCKD / TRUCKTOT

4.2 Calc_pop_emp_60mintt.ave

- User is presented with a list of all layers in “view1” and asked to choose a “netband theme”
- Find a layer in “view1” named “mont_blkgrp.shp”
- Iterate over all the features in the “netband” layer
 - For each feature, use the shape to spatially query the “mont_blkgrp.shp” layer
 - Iterate over the selected features in “mont_blkgrp.shp” layer
 - For each selected feature, prorate the values in the POPULATION and EMPLOYMENT fields based on the percentage of area that actually intersected the “netband” polygon
 - Sum all the prorated POPULATION values in the selection
 - Sum all the prorated EMPLOYMENT values in the selection
 - Set the value of the POP field in the “netband” layer to the summed, prorated POPULATION values
 - Set the value of the EMP field in the “netband” layer to the summed, prorated EMPLOYMENT values

4.3 Calculats_max_pot_gro_final.ave

- Find the following tables in the ArcView project. They are related on County name (REGION field)
 - Eps_vars_lq.dbf (**LQ**)
 - Eps_vars_baseemp.dbf (**BASE**)
 - Eps_vars_90emp.dbf (**90**)
 - Eps_vars_gro.dbf (**GRO**)
- Also find the Maxpotgro.dbf (**FINAL**) table. The **FINAL** table is related to the other 4 tables on the **COUNTY** field. The fields in all the tables are also related to each other on the number appended to the field name. For example, **SIC1** in **FINAL** relates to **LQSIC1** in **LQ**, **GROSIC1** in **GRO**, **EMP90_1** in **90**, and **BASEEMP_1** in **BASE**.
- Iterate over the records in the **FINAL** table
 - Get the related records from the **LQ**, **BASE**, **90** and **GRO** tables (relating on County)
 - Also get the records in the **LQ** and **GRO** tables for the **US**, **5REG**, **MT** regions
 - Iterate over the **SIC** fields in the **FINAL** table
 - For each field, get the value of the related field name in the related record in the other 4 tables
 - Also get the values of the **US**, **5REG** and **MT** fields in the related record in the **LQ** and **GRO** tables
 - Calculate **LQ_GRO**
 - If $LQVAL < 0.0009$ or $LQVAL > REGLQVAL$, $LG_GRO = 0$
 - Else $LG_GRO = (REGLQVAL - LQVAL) * TOTVAL$
 - Calculate **TREND_GRO**
 - $TREND_GRO = ((2025 - 2000) / 10) * (GROVAL - 0.05) * BASEVAL$
 - Calculate **SS_GRO_US**
 - If $USGROVAL < GROVAL$, $SS_GRO_US = 0$
 - Else $SS_GRO_US = ((2025 - 2000) / 10) * (USGROVAL - GROVAL) * BASEVAL$
 - Calculate **SS_GRO_5REG**
 - If $REGGROVAL < GROVAL$, $SS_GRO_5REG = 0$
 - Else $SS_GRO_5REG = ((2025 - 2000) / 10) * (REGGROVAL - GROVAL) * BASEVAL$
 - Calculate **SS_GRO**
 - If $SS_GRO_US \geq SS_GRO_5REG$, $SS_GRO = SS_GRO_US$
 - Else $SS_GRO = SS_GRO_5REG$
 - Calculate **MAX_POT_GRO**
 - If **SicNum** exists in the list (1,2,8,9,10,12,13,14), calculate some values
 - theMax = the maximum of **LQ_GRO**, **TREND_GRO**, **SS_GRO** and zero

- Calculate MAX_POT_GRO
 - If theMax < (1.25 * TREND_GRO), MAX_POT_GRO = theMax
 - Else MAX_POT_GRO = the minimum of (1.25 * TREND_GRO) and theMax
- Else **SicNum** doesn't exist in list
 - theMax = the maximum of LQ_GRO, TREND_GRO, SS_GRO
 - if theMax = TREND_GRO, increment CNTTREND by 1
 - else if theMax = LQ_GRO, increment CNTLQ by 1
 - else if theMax = SS_GRO, increment CNTGRO by 1
 - **MAX_POT_GRO** = theMax
 - If **MAX_POT_GRO** = 0, increment CNTZERO by 1
 - Increment CNT by 1
- If **MAX_POT_GRO** < 0, increment CNTNEG by 1
- Set the value of the current row and the current field to **MAX_POT_GRO**

4.4 Gets_popandemp_innetbands.ave

- Find a layer in "view1" named "block_groups_mt.shp"
- User is presented with a multi-select list of all layers in "view1" and asked to choose a "netband theme"
- Iterate over the selected "themes"
 - For each theme, get the theme name
 - Extract the County name from the theme name
 - If the theme's field list does not contain the field "NAME", then create it (Text, 30)
 - If the theme's field list does not contain the field "POP", then create it (Double, 16, 2)
 - If the theme's field list does not contain the field "EMP", then create it (Double, 16, 2)
 - Iterate over the features in the theme
 - Set the "NAME" field value to the County name
 - Get the feature's shape and spatially select the features in "block_groups_mt.shp"
 - Iterate over the selected features in "block_groups_mt.shp" layer
 - For each selected feature, prorate the values in the POP_00 and WORKERS fields based on the percentage of area that actually intersected the "netband" polygon
 - Sum all the prorated POP_00 values in the selection
 - Sum all the prorated WORKERS values in the selection
 - Set the value of the POP field in the current theme to the summed, prorated POP_00 values



- Set the value of the EMP field in the current theme to the summed, prorated WORKERS values

4.5 Calcs_acs.ave

- Find the following tables in the ArcView project. They are related on County name (COUNTY field in first 3 tables, REGION field in 4th table)
 - Mar_pin.dbf (**PIN**)
 - Empgro_acs.dbf (**ACS**)
 - Vars_by_cnty.dbf (**CNTY**)
 - Eps_vars_baseemp.dbf (**BASE**)
- Also find the Vars_by_sic.dbf (**SIC**) table. The **SIC** table is essentially a lookup for the **ACS** table. One record in the **SIC** table for every **SIC** field in the **ACS** table.
- Iterate over the records in **PIN**
 - If the value of the **DELTA** field is not 0, perform the following tasks
 - Get the value of the following fields in the **PIN** table
 - COUNTY
 - D_AIR
 - D_RAIL
 - D_PCEN
 - D_PCEN
 - D_POP1
 - D_POP2
 - D_ALF
 - D_EMP
 - Get the related value of the **REL** field in the **CNTY** table
 - Get the related value of the **SKILL-2** field in the **CNTY** table
 - Iterate over the **SIC** fields in the **ACS** table
 - For each field, get the related record in the **SIC** table
 - Get the related value of the "**BASEEMP_**" + **SIC#** field from the **BASE** table
 - Calculate **EMPGRO_ACS**
 - $AIRGRO = D_AIR * BASEEMP * TSA_A * (1/REL) * MT_AIR_SEN$
 - $RAILGRO = D_RAIL * BASEEMP * TSA_R * (1/REL) * MT_RAIL_SE$
 - $HIGHGRO = D_PCEN * BASEEMP * (1 - PERC_CAN) * TSA_H * (1/REL) * MT_HIGH_SE$
 - $HIGHGROCAN = D_PCEN * BASEEMP * PERC_CAN * TSA_H * (1/REL) * (1-TARIFF1) * TARIFF2$
 - $CUSTGRO = D_POP1 * BASEEMP * RESBASE * (0.1 + PASSTHROUG)$
 - $LMGRO = D_ALF * BASEEMP * WORKBASE * (1/REL) * ((MATCH_LOW * SKILL_1) + (MATCH_HIGH * SKILL_1 * SKILL_2))$
 - $SUPPACCGRO = D_EMP * BASEEMP * SUPP_HIGH + ((D_AIR * SUPP_AIR) + (D_RAIL * SUPP_RAIL)) * (1+SUPP_SENS)$



- **EMPGRO_ACS** = AIRGRO + RAILGRO + HIGHGRO + HIGHGROCAN + CUSTGRO + LMGRO + SUPPACCGRO
- Set the value of the **SIC** field for the related row in the **ACS** table to **EMPGRO_ACS**

4.6 Calcs_dgo.ave

- Find the following tables in the ArcView project. They are related on County name (COUNTY field)
 - Empgro_acs.dbf (**ACS**)
 - Maxpotgro.dbf (**EPS**)
 - Empgro_dgo.dbf (**DGO**)
- Iterate over the records in the **DGO** table
 - Get the related record from the **ACS** and **EPS** tables
 - Iterate over the **SIC** fields in the **DGO** table
 - Get the **ACS** related record's value of the field with the same name
 - Get the **EPS** related record's value of the field with the same name
 - Set the value of the **SIC** field in the **DGO** table to the minimum of the **ACS** and **EPS** values

4.7 Cao_ave

- Runs on a list of Scenarios provided in a text file
- Add the following tables in the ArcView project.
 - Accessresults.dbf (**PIN**) : relates on COUNTY field
 - Cao_io2.dbf (**IO**) : records related on SIC to fields in DGO table where field name is "SIC" + SIC# and to fields in BASE table where field name is "BASEEMP_" + SIC#
 - Cao_vars.dbf (**CAOVARS**) : relates on SIC field
 - Empgro_cao.dbf (**EGCAO2**) : relates on COUNTY field
 - Empgro_caoinit.dbf (**EGCAO1**) : relates on COUNTY field
 - Empgro_dgo.dbf (**DGO**) : relates on COUNTY field
 - Eps_vars_baseemp.dbf (**BASE**) : relates on REGION field
 - Eps_vars_gro.dbf (**GRO**) : relates on REGION field
 - Eps_vars_lq.dbf (**LQ**) : relates on REGION field
- Create the output table "busat_emp_final.dbf" (**CAO**) with the following fields. This table relates to the other tables on the **SIC** field.
 - SIC (double 8,0)
 - BUSAT1 (double 16,4)
 - BUSAT2 (double 16,4)
 - BUSAT3 (double 16,4)
 - BUSAT4 (double 16,4)
 - BUSAT5 (double 16,4)
 - SICSUM (double 16,4)



- Iterate over the records in the **IO** table
 - Get the value of the **SIC** field
 - Create a new record for the **CAO** table
 - Set the value of the **CAO's SIC** field to the **SIC** field from the **IO** table
- Zero out all the **SIC#** fields for all rows in the **EGCAO1** and **EGCAO2** tables
- Calculate **GRO_DGO_MT** values per **SIC** value and store in a dictionary (**GRO_DGO_MT_DICT**)
 - Iterate over the records in the output **CAO** table
 - Get the **SIC#**
 - Calculate the **MT_GRO_DGO**
 - Sum all the values of the "**SIC**" + **SIC#** field in **DGO** table
 - Get the value of the "**BASEEMPMT**" field in the **CAOVARS** table
 - Calculate **GRO_DGO_MT** = **MT_GRO_DGO** / **BASEEMPMT**
 - Add to dictionary with key of **SIC** and value of **GRO_DGO_MT**
- Iterate over the records in **EGCAO1**
 - Iterate over the **SIC** fields in **EGCAO1**
 - Get the value of the **SIC** field in the **DGO** table for the related record
 - Get the values of the **DAMP1**, **ST_DAMP1** and **TGRO_MT** fields in the related record in the **CAOVARS** table
 - Get the related records in the **LQ**, **BASE** and **GRO** tables
 - Get the value of the **5REG** field in the **LQ** table
 - Get the values of the **US**, **5REG** and **MT** fields in the **GRO** table
 - Get the **IO** record related to the current **SIC** field number
 - Get the related values from the corresponding **SIC** field names in the **LQ**, **GRO**, **BASE** and **IO** tables
 - Get the value from the corresponding **SIC** field name in the **LQ** table for the **5REG** record
 - Get the related value from the **TOEMP** field in the **BASE** table
 - Get the values from the corresponding **SIC** field name in the **GRO** table for the **5REG** and **US** records
 - Calculate **LQ_GRO**
 - If $LQVAL < 0.0009$ or $LQVAL > REG LQVAL$, $LG_GRO = 0$
 - Else $LG_GRO = (REG LQVAL - LQVAL) * TOTEMP$
 - Calculate **TREND_GRO**
 - $TREND_GRO = ((2025 - 2000) / 10) * (GROVAL - 0.05) * BASEVAL$
 - Calculate **SS_GRO_US**
 - If $USGROVAL < GROVAL$, $SS_GRO_US = 0$
 - Else $SS_GRO_US = ((2025 - 2000) / 10) * (USGROVAL - GROVAL) * BASEVAL$
 - Calculate **SS_GRO_5REG**
 - If $REGGROVAL < GROVAL$, $SS_GRO_5REG = 0$

- Else $SS_GRO_5REG = ((2025 - 2000) / 10) * (REGGROVAL - GROVAL) * BASEVAL$
 - Calculate **SS_GRO**
 - If $SS_GRO_US \geq SS_GRO_5REG$, $SS_GRO = SS_GRO_US$
 - Else $SS_GRO = SS_GRO_5REG$
 - Calculate **EMPGRO_CAO_INIT**
 - Get the **GRO_DGO_MT** value from the **GRO_DGO_MT_DICT** on the current **SIC** value
 - $EMPGRO_CAO_RETADJ = EMPGRO_DGO * (1 - DAMP1 * ST_DAMP1)$
 - If $(TREND_GRO < SS_GRO)$ or $(TREND_GRO < LQ_GRO)$,
 $EMPGRO_CAO_INIT = EMPGRO_CAO_RETADJ$
 - Elseif $(GRO_DGO_MT < TREND_GRO_MT)$, $EMPGRO_CAO_INIT = EMPGRO_CAO_RETADJ$
 - Elseif $(TREND_GRO_MT > 0)$, $EMPGRO_CAO_INIT = EMPGRO_CAO_RETADJ * (TREND_GRO_MT / GRO_DGO_MT)$
 - Else $EMPGRO_CAO_INIT = EMPGRO_CAO_RETADJ * 0.5$
 - Set the value of the current field for the current record in the **EGCAO1** table to **EMPGRO_CAO_INIT**
- Iterate over the records in **EGCAO2**
 - Get the value of the **REMI** field
 - Iterate over the **SIC** fields in the **EGCAO2** table
 - Get the value of the same field in the related record in **EGCAO1**
 - Get the value of the **D_EMP** field in the related record in the **PIN** table
 - If $(D_EMP < 0.2)$, $EMPGRO_CAO = EMPGRO_CAO_LINK * (1 + D_EMP) * (1 + 0.000283)$
 - Else $EMPGRO_CAO = EMPGRO_CAO_LINK * (1.2) * (1 + 0.000283)$
 - Set the value of the current field for the current row to **EMPGRO_CAO**
- Iterate over the records in the **CAOVARS** table
 - Get the current record's **SIC#**
 - Iterate over a hard-coded list of regions {1,2,3,4,5}
 - Select rows in **EGCAO2** where **REMI** = current region number
 - Sum the values of the corresponding "**SIC**" + **SIC#** field in the selection in **EGCAO2**
 - Set the value of the "**EMPGRO**" + **REGION#** field in **CAOVARS** to the sum
- Iterate over the records in the output **CAO** table
 - Get the related record in the **CAOVARS** table
 - Multiply the region values by 1.5
 - $regsh1 = regsh1 * 1.5$
 - $regsh2 = regsh2 * 1.5$
 - $regsh3 = regsh3 * 1.5$
 - $regsh4 = regsh4 * 1.5$



- $\text{regsh5} = \text{regsh5} * 1.5$
- Calculate offsets
 - $\text{offset1} = (\text{empgro2} + \text{empgro3} + \text{empgro4} + \text{empgro5}) * (\text{regsh1} * -1)$
 - $\text{offset2} = (\text{empgro1} + \text{empgro3} + \text{empgro4} + \text{empgro5}) * (\text{regsh2} * -1)$
 - $\text{offset3} = (\text{empgro1} + \text{empgro2} + \text{empgro4} + \text{empgro5}) * (\text{regsh3} * -1)$
 - $\text{offset4} = (\text{empgro1} + \text{empgro2} + \text{empgro3} + \text{empgro5}) * (\text{regsh4} * -1)$
 - $\text{offset5} = (\text{empgro1} + \text{empgro2} + \text{empgro3} + \text{empgro4}) * (\text{regsh5} * -1)$
- Set the BUSAT_EMP_FINAL_ field values in the **CAO** table
 - $\text{BUSAT1} = \text{empgro1} + \text{offset1}$
 - $\text{BUSAT2} = \text{empgro2} + \text{offset2}$
 - $\text{BUSAT3} = \text{empgro3} + \text{offset3}$
 - $\text{BUSAT4} = \text{empgro4} + \text{offset4}$
 - $\text{BUSAT5} = \text{empgro5} + \text{offset5}$

4.8 Accessibility Avenue Script Functionality

Stacia Taggart
August 22, 2012

The Avenue scripts were built to support Network Analyst functionality that was lacking in the ArcGIS 8.3/9 version of the extension. The Avenue functions were used as part of the following HEAT tools:

- Assignment
- Accessibility

Purpose

The purpose of the Accessibility analysis is to determine the changes in travel times from each HEAT Zone (County) centroid to various facilities and markets. These facilities and markets include Large Cities, Airports, Border Crossings, Grain Elevators, etc. Changes in accessibility are inputs to the business attraction analysis module.

Process

Add data to the project

- Delete “view1” if it exists
- Create a new blank “view1”
- Add the following tables and layers to the project. Find path to <SCENARIO> from a text file (thepath.txt).
 - Mar_pin.dbf table (**MARPIN**)
 - Commercial_airports.shp (**AIR**)
 - Major_grain_elevators.shp (**GE**)



- Major_border_crossings.shp (**BORDER**)
- Intermodal_terminals.shp (**INTTERM**)
- Access_cities2.shp (**CITY**)
- Countyendpoints.shp (**COUNTY**)
- **SCENARIO** polyline layer
- **SCENARIONODES** point layer
- **NOBUILD** polyline layer
- **NOBUILDNODES** point layer
- Block_groups_mt2.shp (**BLOCK**)

- Save the project

Create PIN Tables

- Pin_closestlandmarkstocounty.dbf (**PINLANDMARK**)
 - Create empty table and add the following fields:

- County (text: 25)
- ScNodeId (long)
- NbNodeId (long)
- AirLocId (text: 25)
- RName (text: 25)
- CrossName (text: 25)
- GeName (text: 25)
- Scen_Air (long)
- Scen_Cro (long)
- Scen_Gel (long)
- Scen_Rail (long)
- Nb_Air (long)
- Nb_Cro (long)
- Nb_Gel (long)
- Nb_Rail (long)
- AirX (double: 16,8)
- AirY (double: 16,8)
- CroX (double: 16,8)
- CroY (double: 16,8)
- RailX (double: 16,8)
- RailY (double: 16,8)
- GelX (double: 16,8)
- GelY (double: 16,8)
- ScAirTime (double: 16,6)
- ScCroTime (double: 16,6)

Node Id Fields

Landmark String
Fields

Landmark Id
Fields

Landmark XY
Fields



- ScGelTime (double: 16,6)
 - ScRaiTime (double: 16,6)
 - NbAirTime (double: 16,6)
 - NbCroTime (double: 16,6)
 - NbGelTime (double: 16,6)
 - NbRaiTime (double: 16,6)
- Add to project
- Pin_closestcitystocounty.dbf (**PINCITY**)
 - Create empty table and add the following fields:
 - County (text: 25)
 - ScNodeId (long)
 - NbNodeId (long)
 - City1 (text: 40)
 - City2 (text: 40)
 - City3 (text: 40)
 - City4 (text: 40)
 - City5 (text: 40)
 - Cpop1 (long)
 - Cpop2 (long)
 - Cpop3 (long)
 - Cpop4 (long)
 - Cpop5 (long)
 - Nbc1 (long)
 - Nbc2 (long)
 - Nbc3 (long)
 - Nbc4 (long)
 - Nbc5 (long)
 - Scc1 (long)
 - Scc2 (long)
 - Scc3 (long)
 - Scc4 (long)
 - Scc5 (long)
 - ScPopCent (double: 16,3)
 - NbPopCent (double: 16,3)
 - Add to project
- Populate COUNTY and NODEID fields in both **PIN** tables
 - Iterate over records in the **MARPIN** table
 - For each record, add a new record to each **PIN** table
 - Set the value of the **COUNTY** field in the **PIN** tables with the value of the **COUNTY** field in the **MARPIN** table
 - Set the value of the **NBNODEID** field in the **PIN** tables with the value of the **TAZID** field in the **MARPIN** table

Time Fields

Node Id Fields

City Fields

Pop Node Fields

Pop Cent Fields



- Set the value of the **SCNODEID** field in the **PIN** tables with the value of the **TAZID** field in the **MARPIN** table

Find Landmarks Closest to NOBUILD scenario

- For each of the **AIR**, **GE**, **BORDER**, **INTTERM** and **CITY** layers,
 - If the **NBNODEID** field does not exist, add it
 - Iterate over the features in the layer
 - Get the feature shape and spatially select the **NOBUILDNODES** layer with a 0.5 buffer
 - If no **NOBUILDNODES** features were selected, select again with a 4000 buffer
 - If no **NOBUILDNODES** features were selected, select again with a 15000 buffer
 - If no **NOBUILDNODES** features were selected, select again with a 1000000 buffer
 - If there are selected features, find the closest feature and get its **NODEID**
 - Otherwise **NODEID** = 0
 - Set the value of the **NBNODEID** field in the layer to the **NODEID**

Find Landmarks Closest to the Input SCENARIO

- For each of the **AIR**, **GE**, **BORDER**, **INTTERM** and **CITY** layers,
 - If the **SCNODEID** field does not exist, add it
 - Iterate over the features in the layer
 - Get the feature shape and spatially select the **SCENARIO** layer with a 0.5 buffer
 - If no **SCENARIO** features were selected, select again with a 4000 buffer
 - If no **SCENARIO** features were selected, select again with a 15000 buffer
 - If no **SCENARIO** features were selected, select again with a 1000000 buffer
 - If there are selected features, find the closest feature and get its **NODEID**
 - Otherwise **NODEID** = 0
 - Set the value of the **SCNODEID** field in the layer to the **NODEID**

Get 5 Closest Cities

- Iterate over the **PINCITY** table records
 - For each record, get the county name and query the **COUNTY** layer on county name
 - Get the point shape from the **COUNTY** layer
 - Iterate over the features in the **CITY** layer
 - Get the 5 closest city features to this county point
 - Sort cities from closest to farthest
 - City1 is closest
 - City5 is farthest





- Populate the **CITY#**, **CPOP#**, **NBC#** and **SCC#** fields in the **PINCITY** record with the corresponding values from the 5 closest cities in the **CITY** layer

Get Closest Airport for each County

- Iterate over the **COUNTY** layer records
 - For each record, get the county name and query the **PINLANDMARK** layer on county name
 - Get the point shape from the **COUNTY** layer
 - Iterate over the features in the **AIR** layer
 - Get the closest airport feature to this county point
 - Populate the following fields in the selected **PINLANDMARK** record with the corresponding values in the closest **AIR** record:
 - **AIRLOCID** = LOCID in AIR record
 - **SCEN_AIR** = SCNODEID in AIR record
 - **NB_AIR** = NBNODEID in AIR record
 - **AIRX** = AIR record shape, X-coordinate
 - **AIRY** = AIR record shape, Y-coordinate

Get Closest Intermodal Terminal for each County

- Iterate over the **COUNTY** layer records
 - For each record, get the county name and query the **PINLANDMARK** layer on county name
 - Get the point shape from the **COUNTY** layer
 - Iterate over the features in the **INTTERM** layer
 - Get the closest Intermodal Terminal feature to this county point
 - Populate the following fields in the selected **PINLANDMARK** record with the corresponding values in the closest **INTTERM** record:
 - **RNAME** = NAME in INTTERM record
 - **SCEN_RAIL** = SCNODEID in INTTERM record
 - **NB_RAIL** = NBNODEID in INTTERM record
 - **RAILX** = INTTERM record shape, X-coordinate
 - **RAILY** = INTTERM record shape, Y-coordinate

Get Closest Grain Elevator for each County

- Iterate over the **COUNTY** layer records
 - For each record, get the county name and query the **PINLANDMARK** layer on county name
 - Get the point shape from the **COUNTY** layer
 - Iterate over the features in the **GE** layer
 - Get the closest Grain Elevator feature to this county point



- Populate the following fields in the selected **PINLANDMARK** record with the corresponding values in the closest **GE** record:
 - **GENAME** = GELEV in GE record
 - **SCEN_GEL** = SCNODEID in GE record
 - **NB_GEL** = NBNODEID in GE record
 - **GELX** = GE record shape, X-coordinate
 - **GELY** = GE record shape, Y-coordinate

Get Closest Border Crossing for each County

- Iterate over the **COUNTY** layer records
 - For each record, get the county name and query the **PINLANDMARK** layer on county name
 - Get the point shape from the **COUNTY** layer
 - Iterate over the features in the **BORDER** layer
 - Get the closest Border Crossing feature to this county point
 - Populate the following fields in the selected **PINLANDMARK** record with the corresponding values in the closest **BORDER** record:
 - **CROSSNAME** = DESCRIP in BORDER record
 - **SCEN_CRO** = SCNODEID in BORDER record
 - **NB_CRO** = NBNODEID in BORDER record
 - **CROX** = BORDER record shape, X-coordinate
 - **CROY** = BORDER record shape, Y-coordinate

Get Travel Times for PIN Cities and Landmarks (NOBUILD)

- Using Network Analyst, make a network from the **NOBUILD** layer
- Set the “**TRAVELTIME**” field as the COST field

Calculate NOBUILD Travel Times to Landmarks

- **PINLANDMARK** table is related to **NOBUILDNODES** (NBNODEID to NODEID)
- Iterate over records in **PINLANDMARK** table
 - Get the related **NOBUILDNODES** point shape
 - Create an Airport point from the **AIRX** and **AIRY** fields
 - Create a Border Crossing point from the **CROX** and **CROY** fields
 - Create an Intermodal Terminal point from the **RAILX** and **RAILY** fields
 - Create a Grain Elevator point from the **GELX** and **GELY** fields
 - Using the **NOBUILD** network layer, calculate the times for the following routes:
 - Current **NOBUILDNODES** point to the Airport point (**AIRTIME**)
 - Current **NOBUILDNODES** point to the Border Crossing point (**CROTIME**)
 - Current **NOBUILDNODES** point to the Intermodal Terminal point (**TERMTIME**)



- Current **NOBUILDNODES** point to the Grain Elevator point (**GETIME**)
- Set the following field values in the current **PINLANDMARK** record:
 - **NBAIRTIME** = **AIRTIME**
 - **NBCROTIME** = **CROTIME**
 - **NBRAITIME** = **TERMTIME**
 - **NBGELTIME** = **GETIME**

Calculate **NOBUILD** Travel Times to Cities

- **PINCITY** table is related to **NOBUILDNODES** (**NBNODEID** to **NODEID**)
- Iterate over records in **PINCITY** table
 - Get the related **NOBUILDNODES** point shape
 - Get the City NodeId values
 - **NBC1**, **NBC2**, **NBC3**, **NBC4**, **NBC5**
 - Get the related **NOBUILDNODES** point shapes for each City NodeId
 - Using the **NOBUILD** network layer, calculate the times for the following routes:
 - Current **NOBUILDNODES** point to the City1 point (**C1TIME**)
 - Current **NOBUILDNODES** point to the City2 point (**C2TIME**)
 - Current **NOBUILDNODES** point to the City3 point (**C3TIME**)
 - Current **NOBUILDNODES** point to the City4 point (**C4TIME**)
 - Current **NOBUILDNODES** point to the City5 point (**C5TIME**)
 - Get the values of the **CPOP** fields in the **PINCITY** record
 - Make the following calculations:
 - $c1val = (popc1 / (c1time ^ 2))^{0.5}$
 - $c2val = (popc2 / (c2time ^ 2))^{0.5}$
 - $c3val = (popc3 / (c3time ^ 2))^{0.5}$
 - $c4val = (popc4 / (c4time ^ 2))^{0.5}$
 - $c5val = (popc5 / (c5time ^ 2))^{0.5}$
 - Set the value of the **NBPOPCEINT** field for the current record in the **PINCITY** table to $c1val + c2val + c3val + c4val + c5val$

Create Accessibility Polygons (**NOBUILD**)

- Create a new empty polygon feature class called **NOBUILDPGONS.SHP**
- Add a field named **SITE_ID** (long)
- Select nodes in **NOBUILDNODES** where $(([nodeid] < 999) \text{ and } ([district] <> 99))$
- Iterate over selected nodes
 - For each node, use Network Analyst to create a 60-minute drive time polygon
 - Create a new record in **NOBUILDPGONS**
 - Add the polygon into the record



- Set the **SITE_ID** field value to the **NODEID** from the current feature in **NOBUILDNODES**

Get Travel Times for PIN Cities and Landmarks (SCENARIO)

- Using Network Analyst, make a network from the **SCENARIO** layer
- Set the “**TRAVELTIME**” field as the **COST** field

Calculate SCENARIO Travel Times to Landmarks

- **PINLANDMARK** table is related to **SCENARIO_NODES** (**SCNODEID** to **NODEID**)
- Iterate over records in **PINLANDMARK** table
 - Get the related **SCENARIO_NODES** point shape
 - Create an Airport point from the **AIRX** and **AIRY** fields
 - Create a Border Crossing point from the **CROX** and **CROY** fields
 - Create an Intermodal Terminal point from the **RAILX** and **RAILY** fields
 - Create a Grain Elevator point from the **GELX** and **GELY** fields
 - Using the **SCENARIO** network layer, calculate the times for the following routes:
 - Current **SCENARIO_NODES** point to the Airport point (**AIRTIME**)
 - Current **SCENARIO_NODES** point to the Border Crossing point (**CROTIME**)
 - Current **SCENARIO_NODES** point to the Intermodal Terminal point (**TERMTIME**)
 - Current **SCENARIO_NODES** point to the Grain Elevator point (**GETIME**)
 - Set the following field values in the current **PINLANDMARK** record:
 - **SCAIRTIME** = **AIRTIME**
 - **SCCROTIME** = **CROTIME**
 - **SCRAITIME** = **TERMTIME**
 - **SCGELTIME** = **GETIME**

Calculate SCENARIO Travel Times to Cities

- **PINCITY** table is related to **SCENARIO_NODES** (**SCNODEID** to **NODEID**)
- Iterate over records in **PINCITY** table
 - Get the related **SCENARIO_NODES** point shape
 - Get the City NodeId values
 - **SCC1**, **SCC2**, **SCC3**, **SCC4**, **SCC5**
 - Get the related **SCENARIO_NODES** point shapes for each City NodeId
 - Using the **SCENARIO** network layer, calculate the times for the following routes:
 - Current **SCENARIO_NODES** point to the City1 point (**C1TIME**)
 - Current **SCENARIO_NODES** point to the City2 point (**C2TIME**)
 - Current **SCENARIO_NODES** point to the City3 point (**C3TIME**)
 - Current **SCENARIO_NODES** point to the City4 point (**C4TIME**)



- Current **SCENARIO_NODES** point to the City5 point (**C5TIME**)
- Get the values of the **CPOP** fields in the **PINCITY** record
- Make the following calculations:
 - $c1val = (popc1 / (c1time ^ 2))^{0.5}$
 - $c2val = (popc2 / (c2time ^ 2))^{0.5}$
 - $c3val = (popc3 / (c3time ^ 2))^{0.5}$
 - $c4val = (popc4 / (c4time ^ 2))^{0.5}$
 - $c5val = (popc5 / (c5time ^ 2))^{0.5}$
- Set the value of the **SCPOPCENT** field for the current record in the **PINCITY** table to $c1val + c2val + c3val + c4val + c5val$

Create Accessibility Polygons (SCENARIO)

- Create a new empty polygon feature class called **SCENARIOPGONS.SHP**
- Add a field named **SITE_ID** (long)
- Select nodes in **SCENARIO_NODES** where $(([nodeid] < 999) \text{ and } ([district] < > 99))$
- Iterate over selected nodes
 - For each node, use Network Analyst to create a 60-minute drive time polygon
 - Create a new record in **SCENARIOPGONS**
 - Add the polygon into the record
 - Set the **SITE_ID** field value to the **NODEID** from the current feature in **SCENARIO_NODES**

Calculate Prorated Accessibility Values (NOBUILD)

- Add the following fields to the **NOBUILDPGONS** feature class
 - **NAME** (text: 30)
 - **POP** (double: 16,2)
 - **EMP** (double: 16,2)
 - **LF** (double: 16,2)
 - **NAMENUEW** (text: 18)
- Iterate over the features in the **NOBUILDPGONS** feature class
 - Get the polygon for the current feature
 - Use the polygon to spatially select the **BLOCK** layer
 - Iterate over the selected features in the **BLOCK** layer
 - For each selected feature, prorate the values in the POP, EMP and LF fields based on the percentage of area that actually intersected the **NOBUILDPGONS** polygon
 - Sum all the prorated POP values in the selection
 - Sum all the prorated EMP values in the selection
 - Sum all the prorated LF values in the selection



- Set the value of the POP field in the **NOBUILDPGONS** layer to the summed, prorated POP values
- Set the value of the EMP field in the **NOBUILDPGONS** layer to the summed, prorated EMP values
- Set the value of the LF field in the **NOBUILDPGONS** layer to the summed, prorated LF values

Calculate Prorated Accessibility Values (SCENARIO)

- Add the following fields to the **SCENARIOPGONS** feature class
 - **NAME** (text: 30)
 - **POP** (double: 16,2)
 - **EMP** (double: 16,2)
 - **LF** (double: 16,2)
 - **NAMENew** (text: 18)
- Iterate over the features in the **SCENARIOPGONS** feature class
 - Get the polygon for the current feature
 - Use the polygon to spatially select the **BLOCK** layer
 - Iterate over the selected features in the **BLOCK** layer
 - For each selected feature, prorate the values in the POP, EMP and LF fields based on the percentage of area that actually intersected the **SCENARIOPGONS** polygon
 - Sum all the prorated POP values in the selection
 - Sum all the prorated EMP values in the selection
 - Sum all the prorated LF values in the selection
 - Set the value of the POP field in the **SCENARIOPGONS** layer to the summed, prorated POP values
 - Set the value of the EMP field in the **SCENARIOPGONS** layer to the summed, prorated EMP values
 - Set the value of the LF field in the **SCENARIOPGONS** layer to the summed, prorated LF values

Make the Final Accessibility Results Table

- Create a new empty table (**ACCESSRESULTS.DBF**)
- Add the following fields to the table
 - **COUNTY** (text: 40)
 - **NODEID** (long)
 - **D_PCEN** (double: 16,8)
 - **D_AIR** (double: 16,8)
 - **D_RAIL** (double: 16,8)



- D_GELEV(double: 16,8)
- D_PCEN TC (double: 16,8)
- D_POP1(double: 16,8)
- D_EMP (double: 16,8)
- D_ALF (double: 16,8)
- D_POP2 (double: 16,8)
- DELTA (long)
- Create a new empty table (**ACCESSRESULTS2.DBF**)
- Add the following fields to the table
 - COUNTY (text: 40)
 - NODEID (long)
 - POP CENTERS (double: 16,8)
 - AIRPORTS (double: 16,8)
 - RAIL TERM (double: 16,8)
 - GRAIN ELEV (double: 16,8)
 - CANADA (double: 16,8)
 - POPULATION (double: 16,8)
 - EMPLOYMENT (double: 16,8)
 - LABORFORCE (double: 16,8)
 - DELTA (long)
- Iterate over the records in the **PINLANDMARK** table
 - For each record, add a new record to the **ACCESSRESULTS** table
 - Set the value of the **COUNTY** field to value of the **COUNTY** field in the **PINLANDMARK** table
 - Set the value of the **NODEID** field to value of the **NBNODEID** field in the **PINLANDMARK** table
- Iterate over the records in the **ACCESSRESULTS** table
 - Get the current record's NODEID
 - Query the **PINLANDMARK** table with the query string: ([**NBNODEID**]= " + nodeid.asstring + ")
 - Query the **PINCITIES** table with the query string: ([**NBNODEID**]= " + nodeid.asstring + ")
 - Query the **NOBUILDPGONS** table with the query string: ([**SITE_ID**]= " + nodeid.asstring + ")
 - Query the **SCENARIOPGONS** table with the query string: ([**SITE_ID**]= " + nodeid.asstring + ")
 - Iterate over the selected records in the **PINCITIES** table
 - From the **NBPOP CENT** and **SCPOP CENT** fields, calculate **D_PCEN T** = (**SCPOP CENT** – **NBPOP CENT**) / **NBPOP CENT**
 - If **D_PCEN T** is between -0.001 and 0, then default **D_PCEN T** to 0
 - Set the value of the **D_PCEN T** field in the **ACCESSRESULTS** table to **D_PCEN T**
 - Iterate over the selected records in the **PINLANDMARK** table

- From the **NBAIRTIME** and **SCAIRTIME** fields, calculate **D_AIR** = $(\text{NBAIRTIME} - \text{SCAIRTIME}) / \text{SCAIRTIME}$
- If **D_AIR** is between -0.001 and 0, then default **D_AIR** to 0
- Set the value of the **D_AIR** field in the **ACCESSRESULTS** table to **D_AIR**
- From the **NBCROTIME** and **SCCROTIME** fields, calculate **D_PCENCTC** = $(\text{NBCROTIME} - \text{SCCROTIME}) / \text{SCCROTIME}$
- If **D_PCENCTC** is between -0.001 and 0, then default **D_PCENCTC** to 0
- Set the value of the **D_PCENCTC** field in the **ACCESSRESULTS** table to **D_PCENCTC**
- From the **NBGELTIME** and **SCGELTIME** fields, calculate **D_GELEV** = $(\text{NBGELTIME} - \text{SCGELTIME}) / \text{SCGELTIME}$
- If **D_GELEV** is between -0.001 and 0, then default **D_GELEV** to 0
- Set the value of the **D_GELEV** field in the **ACCESSRESULTS** table to **D_GELEV**
- From the **NBRAITIME** and **SCRAITIME** fields, calculate **D_RAIL** = $(\text{NBRAITIME} - \text{SCRAITIME}) / \text{SCRAITIME}$
- If **D_RAIL** is between -0.001 and 0, then default **D_RAIL** to 0
- Set the value of the **D_RAIL** field in the **ACCESSRESULTS** table to **D_RAIL**
- Iterate over the selected features in **NOBUILDPGONS**
 - **NBPOP** = the value of the **POP** field
 - **NBEMP** = the value of the **EMP** field
 - **NBLF** = the value of the **LF** field
- Iterate over the selected features in **SCENARIOPGONS**
 - **ALTPOP** = the value of the **POP** field
 - **ALTEMP** = the value of the **EMP** field
 - **ALTLF** = the value of the **LF** field
- Perform the following calculations:
 - **D_POP** = $(\text{ALTPOP} - \text{NBPOP}) / \text{NBPOP}$
 - **D_EMP** = $(\text{ALTEMP} - \text{NBEMP}) / \text{NBEMP}$
 - **D_ALF** = $(\text{ALTLF} - \text{NBLF}) / \text{NBLF}$
 - If any of these values is between -0.001 and 0, default them to 0
 - If any of these values is greater than 0.5, default them to 0.5
- Set the value of the **D_POP1** field in the **ACCESSRESULTS** table to **D_POP**
- Set the value of the **D_POP2** field in the **ACCESSRESULTS** table to **D_POP**
- Set the value of the **D_ALF** field in the **ACCESSRESULTS** table to **D_ALF**
- Set the value of the **D_EMP** field in the **ACCESSRESULTS** table to **D_EMP**
- Set the Delta values
 - Iterate over the records in the **ACCESSRESULTS** table
 - If any of the values of any of the fields, except **NODEID** and **COUNTY**, is greater than 0, then set the **DELTA** field in the **ACCESSRESULTS** current record to 1
- Ask the user if this is a 4-lane scenario
 - If yes, iterate over the records in the **ACCESSRESULTS** table
 - For each record, iterate over the fields in the **ACCESSRESULTS** table



- For each value in each field, if it's not the **DELTA** or **COUNTY** fields, then multiply the value by 1.5
- If the result is greater than 0.5, the set the value at 0.5
- Set the value of the field in the **ACCESSRESULTS** table to the new value
- Copy all the records from the **ACCESSRESULTS** table to the **ACCESSRESULTS2** table
 - Map fields as follows:
 - COUNTY → COUNTY
 - NODEID → NODEID
 - D_PCEN → POP CENTERS
 - D_AIR → AIRPORTS
 - D_RAIL → RAIL TERM
 - D_GELEV → GRAIN ELEV
 - D_PCEN → CANADA
 - D_POP1 → POPULATION
 - D_EMP → EMPLOYMENT
 - D_ALF → LABORFORCE
 - Note that there is no mapping for **D_POP2**

Comment [SLT3]: Shouldn't we also skip NODEID?